



12

Document Object Model (DOM): Objects and Collections



OBJECTIVES

In this chapter you will:

- Use JavaScript and the W3C Document Object Model to create dynamic web pages.
- Learn the concept of DOM nodes and DOM trees.
- Traverse, edit and modify elements in an HTML5 document.
- Change CSS styles dynamically.
- Create JavaScript animations.



12.1 Introduction

12.2 Modeling a Document: DOM Nodes and Trees

12.3 Traversing and Modifying a DOM Tree

12.4 DOM Collections

12.5 Dynamic Styles

12.6 Using a Timer and Dynamic Styles to Create Animated Effects



12.1 Introduction

- ▶ The Document Object Model gives you scripting access to *a//* the elements on a web page. Using JavaScript, you can create, modify and remove elements in the page dynamically.



Software Engineering Observation 12.1

With the DOM, HTML5 elements can be treated as objects, and many attributes of HTML5 elements can be treated as properties of those objects. Then objects can be scripted with JavaScript to achieve dynamic effects.

12.2 Modeling a Document: DOM Nodes and Trees



- ▶ `getElementById` method
 - Returns objects called DOM nodes
 - *Every* piece of an HTML5 page (elements, attributes, text, etc.) is modeled in the web browser by a DOM node
- ▶ The nodes in a document make up the page's DOM tree, which describes the relationships among elements
- ▶ Nodes are related to each other through child-parent relationships
- ▶ A node can have multiple children, but only one parent
- ▶ Nodes with the same parent node are referred to as siblings
- ▶ The `html` node in a DOM tree is called the root node, because it has no parent

Browser	Command to display developer tools
Chrome	Windows/Linux: <i>Control + Shift + i</i> Mac OS X: <i>Command + Option + i</i>
Firefox	Windows/Linux: <i>Control + Shift + i</i> Mac OS X: <i>Command + Shift + i</i>
Internet Explorer	<i>F12</i>
Opera	Windows/Linux: <i>Control + Shift + i</i> Mac OS X: <i>Command + Option + i</i>
Safari	Windows/Linux: <i>Control + Shift + i</i> Mac OS X: <i>Command + Option + i</i>

Fig. 12.1 | Commands for displaying developer tools in desktop browsers.



```
1  <!DOCTYPE html>
2
3  <!-- Fig. 12.2: domtree.html -->
4  <!-- Demonstration of a document's DOM tree. -->
5  <html>
6      <head>
7          <meta charset = "utf-8">
8          <title>DOM Tree Demonstration</title>
9      </head>
10     <body>
11         <h1>An HTML5 Page</h1>
12         <p>This page contains some basic HTML5 elements. The DOM tree
13             for the document contains a DOM node for every element</p>
14         <p>Here's an unordered list:</p>
15         <ul>
16             <li>One</li>
17             <li>Two</li>
18             <li>Three</li>
19         </ul>
20     </body>
21 </html>
```

Fig. 12.2 | Demonstration of a document's DOM tree. (Part 1 of 3.)

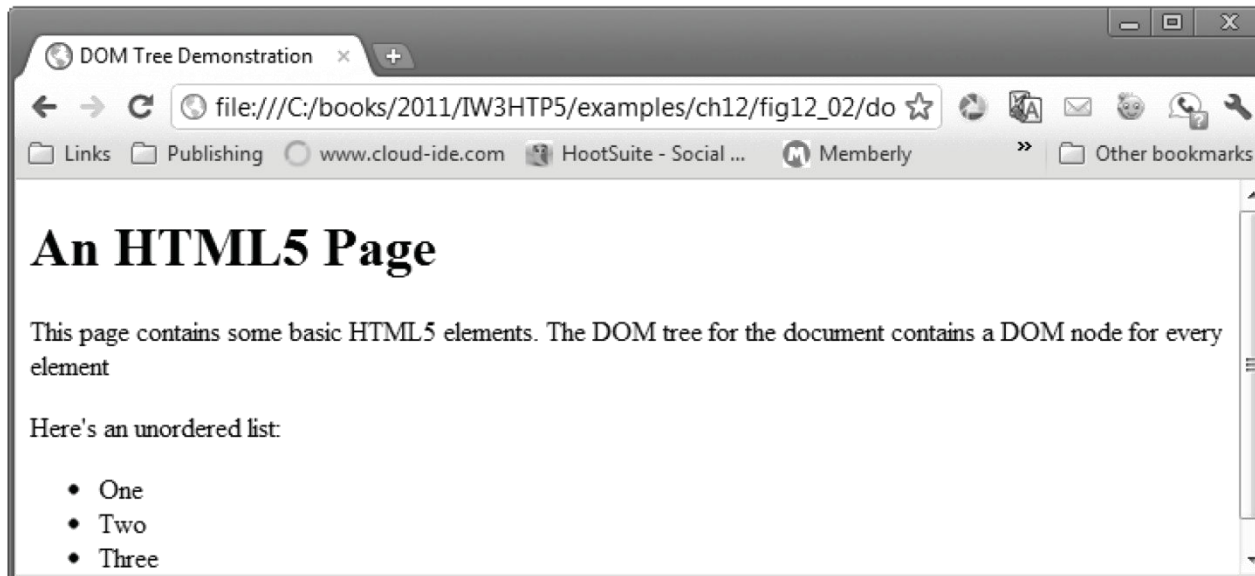


Fig. 12.2 | Demonstration of a document's DOM tree. (Part 2 of 3.)

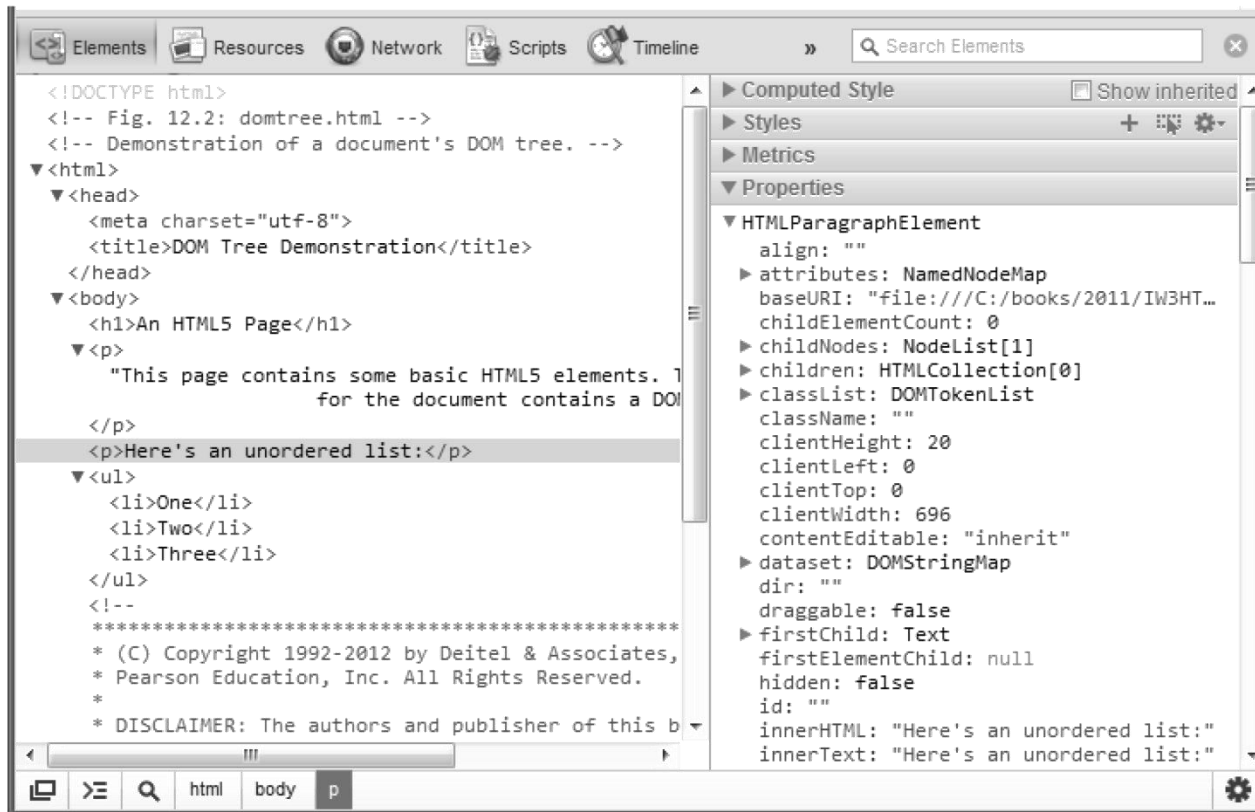


Fig. 12.2 | Demonstration of a document's DOM tree. (Part 3 of 3.)

12.3 Traversing and Modifying a DOM Tree



- ▶ The next example demonstrates several DOM node features and two additional document-object methods.
 - It allows you to highlight, modify, insert and remove elements.
- ▶ CSS class `highlighted` is applied dynamically to elements in the document as we add, remove and select elements using the form.



```
1  /* Fig. 12.3: style.css */
2  /* CSS for dom.html. */
3  h1, h3      { text-align: center;
4                font-family: tahoma, geneva, sans-serif; }
5  p           { margin-left: 5%;
6                margin-right: 5%;
7                font-family: arial, helvetica, sans-serif; }
8  ul          { margin-left: 10%; }
9  a           { text-decoration: none; }
10 a:hover      { text-decoration: underline; }
11 .nav         { width: 100%;
12                border-top: 3px dashed blue;
13                padding-top: 10px; }
14 .highlighted { background-color: yellow; }
15 input        { width: 150px; }
16 form > p     { margin: 0px; }
```

Fig. 12.3 | CSS for basic DOM functionality example.

12.3 Traversing and Modifying a DOM Tree (Cont.)



- ▶ We'll manipulate the HTML5 document dynamically by modifying its DOM.

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 12.4: dom.html -->
4  <!-- Basic DOM functionality. -->
5  <html>
6      <head>
7          <meta charset = "utf-8">
8          <title>Basic DOM Functionality</title>
9          <link rel = "stylesheet" type = "text/css" href = "style.css">
10         <script src = "dom.js"></script>
11     </head>
12     <body>
13         <h1 id = "bigheading" class = "highlighted">
14             [bigheading] DHTML Object Model</h1>
15         <h3 id = "smallheading">[smallheading] Element Functionality</h3>
16         <p id = "para1">[para1] The Document Object Model (DOM) allows for
17             quick, dynamic access to all elements in an HTML5 document for
18             manipulation with JavaScript.</p>
```

Fig. 12.4 | HTML5 document that's used to demonstrate DOM functionality for dynamically adding, removing and selecting elements. (Part I of 4.)



```
19 <p id = "para2">[para2] For more information, check out the
20 "JavaScript and the DOM" section of Deitel's
21 <a id = "link" href = "http://www.deitel.com/javascript">
22 [link] JavaScript Resource Center.</a></p>
23 <p id = "para3">[para3] The buttons below demonstrate:(list)</p>
24 <ul id = "list">
25 <li id = "item1">[item1] getElementById and parentNode</li>
26 <li id = "item2">[item2] insertBefore and appendChild</li>
27 <li id = "item3">[item3] replaceChild and removeChild</li>
28 </ul>
29 <div id = "nav" class = "nav">
30 <form onsubmit = "return false" action = "#">
31 <p><input type = "text" id = "gbi" value = "bigheading">
32 <input type = "button" value = "Get By id"
33 id = "byIdButton"></p>
34 <p><input type = "text" id = "ins">
35 <input type = "button" value = "Insert Before"
36 id = "insertButton"></p>
37 <p><input type = "text" id = "append">
38 <input type = "button" value = "Append Child"
39 id = "appendButton"></p>
```

Fig. 12.4 | HTML5 document that's used to demonstrate DOM functionality for dynamically adding, removing and selecting elements. (Part 2 of 4.)

```
40      <p><input type = "text" id = "replace">
41          <input type = "button" value = "Replace Current"
42              id = "replaceButton()"></p>
43      <p><input type = "button" value = "Remove Current"
44          id = "removeButton"></p>
45      <p><input type = "button" value = "Get Parent"
46          id = "parentButton"></p>
47  </form>
48 </div>
49 </body>
50 </html>
```

Fig. 12.4 | HTML5 document that's used to demonstrate DOM functionality for dynamically adding, removing and selecting elements. (Part 3 of 4.)

The document when it first loads. It begins with the large heading highlighted.

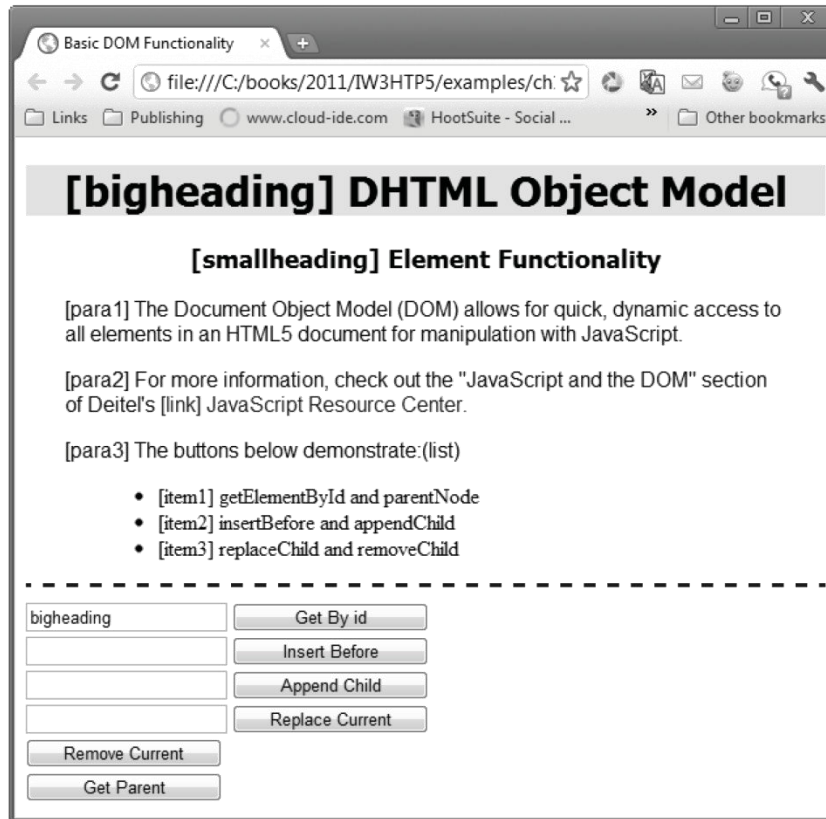


Fig. 12.4 | HTML5 document that's used to demonstrate DOM functionality for dynamically adding, removing and selecting elements (Part 1 of 1)

12.3 Traversing and Modifying a DOM Tree (Cont.)



- ▶ The JavaScript code declares two variables
 - Variable `currentNode` keeps track of the currently highlighted node—the functionality of each button depends on which node is currently selected.
 - Variable `idcount` is used to assign a unique `id` to any new elements that are created.
- ▶ The remainder of the JavaScript code contains event-handling functions for the buttons and two helper functions that are called by the event handlers.



```
1 // Fig. 12.5: dom.js
2 // Script to demonstrate basic DOM functionality.
3 var currentNode; // stores the currently highlighted node
4 var idcount = 0; // used to assign a unique id to new elements
5
6 // register event handlers and initialize currentNode
7 function start()
8 {
```

Fig. 12.5 | Script to demonstrate basic DOM functionality. (Part I of 6.)



```
9    document.getElementById( "byIdButton" ).addEventListener(  
10        "click", byId, false );  
11    document.getElementById( "insertButton" ).addEventListener(  
12        "click", insert, false );  
13    document.getElementById( "appendButton" ).addEventListener(  
14        "click", appendNode, false );  
15    document.getElementById( "replaceButton" ).addEventListener(  
16        "click", replaceCurrent, false );  
17    document.getElementById( "removeButton" ).addEventListener(  
18        "click", remove, false );  
19    document.getElementById( "parentButton" ).addEventListener(  
20        "click", parent, false );  
21  
22    // initialize currentNode  
23    currentNode = document.getElementById( "bigheading" );  
24 } // end function start  
25  
26 // call start after the window loads  
27 window.addEventListener( "load", start, false );  
28
```

Fig. 12.5 | Script to demonstrate basic DOM functionality. (Part 2 of 6.)



```
29 // get and highlight an element by its id attribute
30 function byId()
31 {
32     var id = document.getElementById( "gbi" ).value;
33     var target = document.getElementById( id );
34
35     if ( target )
36         switchTo( target );
37 } // end function byId
38
39 // insert a paragraph element before the current element
40 // using the insertBefore method
41 function insert()
42 {
43     var newNode = createNewNode(
44         document.getElementById( "ins" ).value );
45     currentNode.parentNode.insertBefore( newNode, currentNode );
46     switchTo( newNode );
47 } // end function insert
48
```

Fig. 12.5 | Script to demonstrate basic DOM functionality. (Part 3 of 6.)



```
49 // append a paragraph node as the child of the current node
50 function appendNode()
51 {
52     var newNode = createNewNode(
53         document.getElementById( "append" ).value );
54     currentNode.appendChild( newNode );
55     switchTo( newNode );
56 } // end function appendNode
57
58 // replace the currently selected node with a paragraph node
59 function replaceCurrent()
60 {
61     var newNode = createNewNode(
62         document.getElementById( "replace" ).value );
63     currentNode.parentNode.replaceChild( newNode, currentNode );
64     switchTo( newNode );
65 } // end function replaceCurrent
66
```

Fig. 12.5 | Script to demonstrate basic DOM functionality. (Part 4 of 6.)



```
67 // remove the current node
68 function remove()
69 {
70     if ( currentNode.parentNode == document.body )
71         alert( "Can't remove a top-level element." );
72     else
73     {
74         var oldNode = currentNode;
75         switchTo( oldNode.parentNode );
76         currentNode.removeChild( oldNode );
77     }
78 } // end function remove
79
80 // get and highlight the parent of the current node
81 function parent()
82 {
83     var target = currentNode.parentNode;
84
85     if ( target != document.body )
86         switchTo( target );
87     else
88         alert( "No parent." );
89 } // end function parent
```

Fig. 12.5 | Script to demonstrate basic DOM functionality. (Part 5 of 6.)



```
90
91 // helper function that returns a new paragraph node containing
92 // a unique id and the given text
93 function createNewNode( text )
94 {
95     var newNode = document.createElement( "p" );
96     nodeId = "new" + idcount;
97     ++idcount;
98     newNode.setAttribute( "id", nodeId ); // set newNode's id
99     text = "[" + nodeId + "]" + text;
100     newNode.appendChild( document.createTextNode( text ) );
101     return newNode;
102 } // end function createNewNode
103
104 // helper function that switches to a new currentNode
105 function switchTo( newNode )
106 {
107     currentNode.setAttribute( "class", "" ); // remove old highlighting
108     currentNode = newNode;
109     currentNode.setAttribute( "class", "highlighted" ); // highlight
110     document.getElementById( "gbi" ).value =
111         currentNode.getAttribute( "id" );
112 } // end function switchTo
```

Fig. 12.5 | Script to demonstrate basic DOM functionality. (Part 6 of 6.)

12.3 Traversing and Modifying a DOM Tree (Cont.)



Finding and Highlighting an Element Using `getElementById`, `setAttribute` and `getAttribute`

- ▶ The first row of the form allows the user to enter the id of an element into the text field and click the Get By Id button to find and highlight the element.

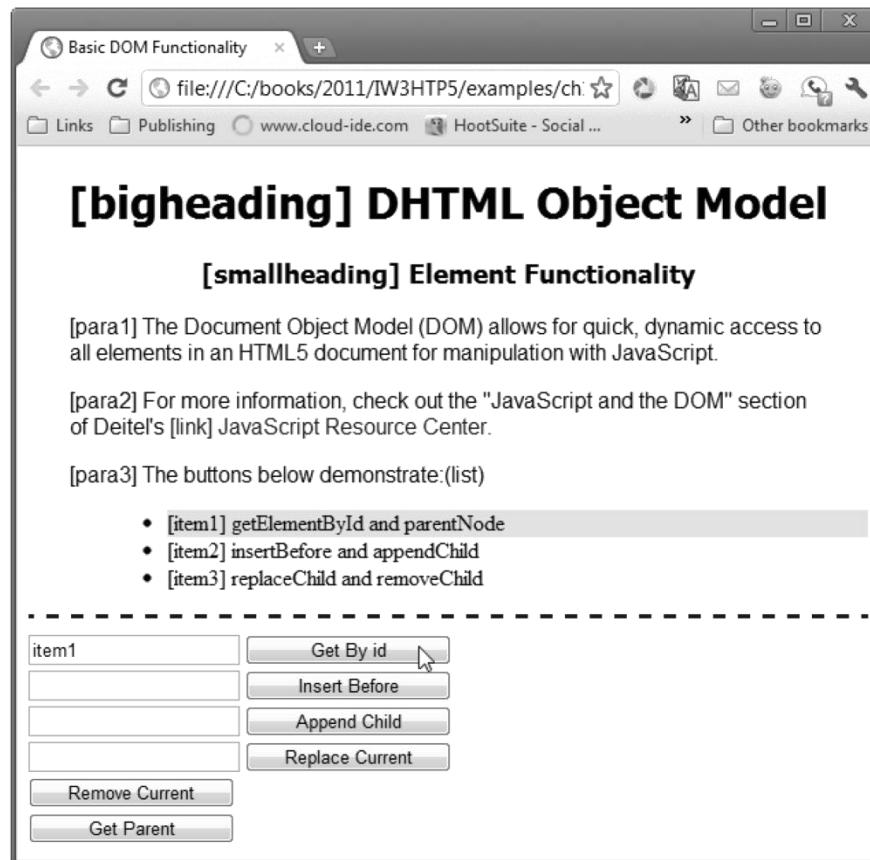


Fig. 12.6 | The document of Figure 12.4 after using the Get By id button to select item1.

12.3 Traversing and Modifying a DOM Tree (Cont.)



- ▶ The DOM element methods `setAttribute` and `getAttribute` allow you to modify an attribute value and get an attribute value, respectively.

12.3 Traversing and Modifying a DOM Tree (Cont.)



- ▶ **document object createElement method**
 - Creates a new DOM node, taking the tag name as an argument. It does not *insert* the element on the page.
- ▶ **document object createTextNode method**
 - Creates a DOM node that contains only text. Given a string argument, createTextNode inserts the string into the text node.
- ▶ **Method appendChild**
 - Inserts a child node (passed as an argument) after any existing children of the node on which it's called
- ▶ **Property parentNode contains the node's parent**
- ▶ **insertBefore method**
 - Inserts newNode as a child of the parent directly before currentNode.
- ▶ **replaceChild method**
 - Receives as its first argument the new node to insert and as its second argument the node to replace.
- ▶ **removeChild method**
 - Remove the oldNode (a child of the new currentNode) from its place in the HTML5 document.

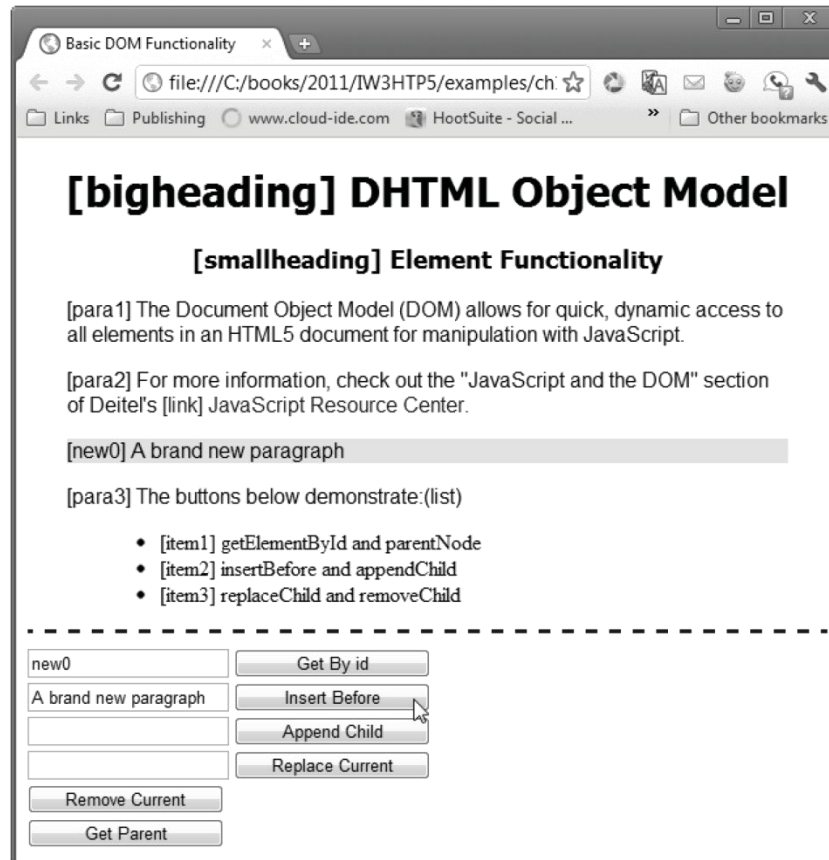


Fig. 12.7 | The document of Figure 12.4 after selecting para3 with the **Get By id** button, then using the **Insert Before** button to insert a new paragraph before para3.

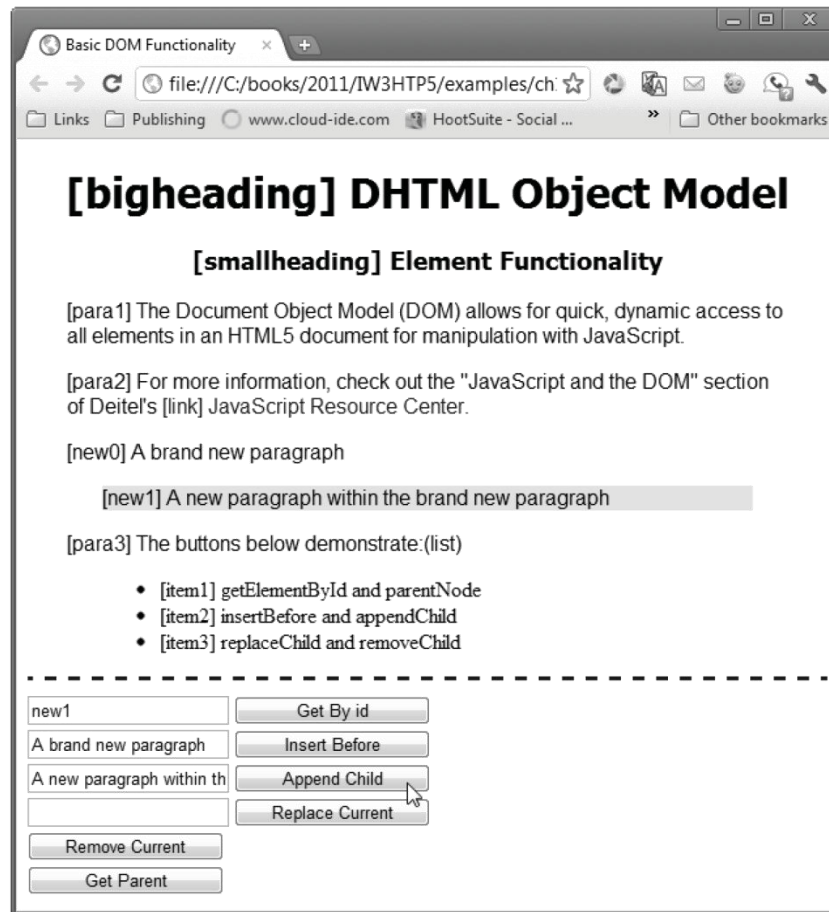


Fig. 12.8 | The document of Figure 12.4 after using the **Append Child** button to append a child to the new paragraph in Figure 12.7.

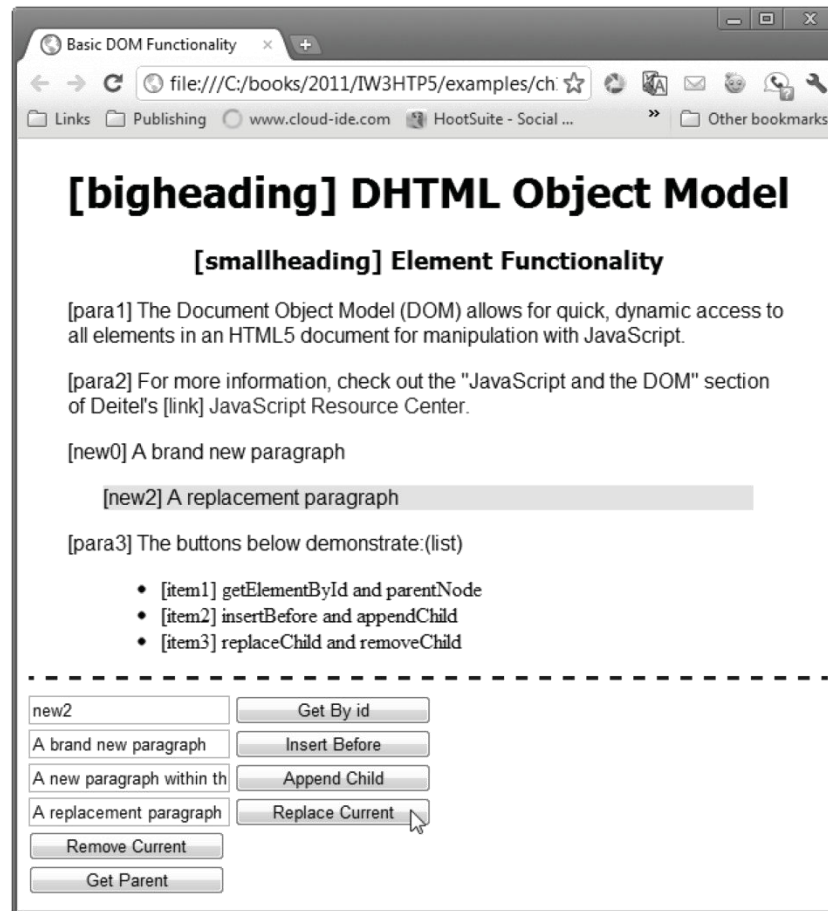


Fig. 12.9 | The document of Figure 12.4 after using the Replace Current button to replace the paragraph created in Figure 12.8.

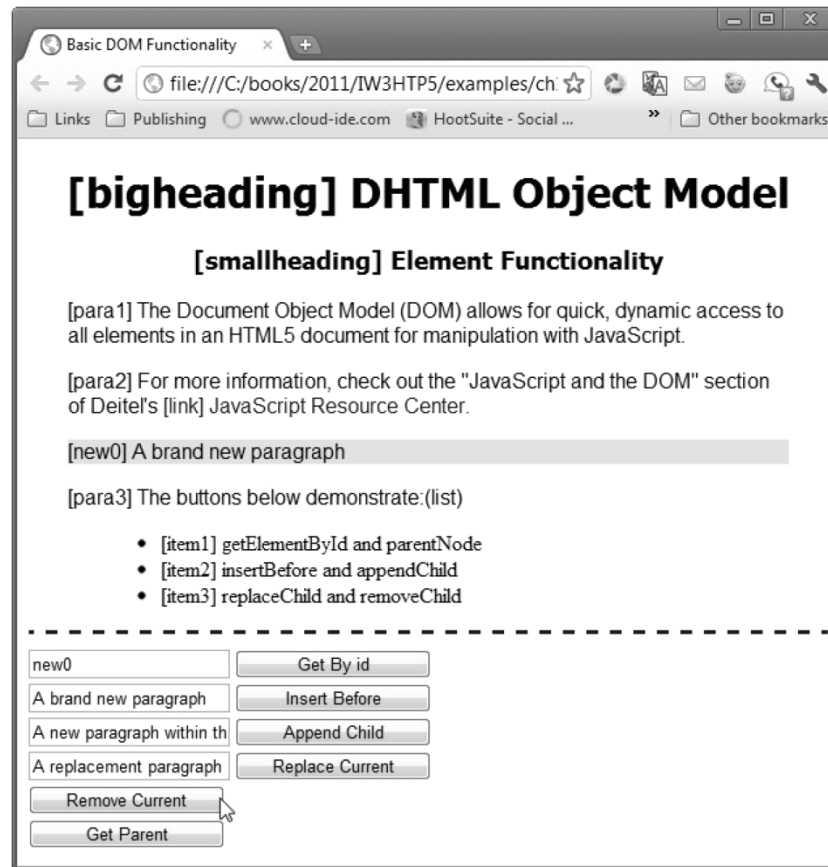


Fig. 12.10 | The document of Figure 12.4 after using the **Remove Current** button to remove the paragraph highlighted in Figure 12.9.

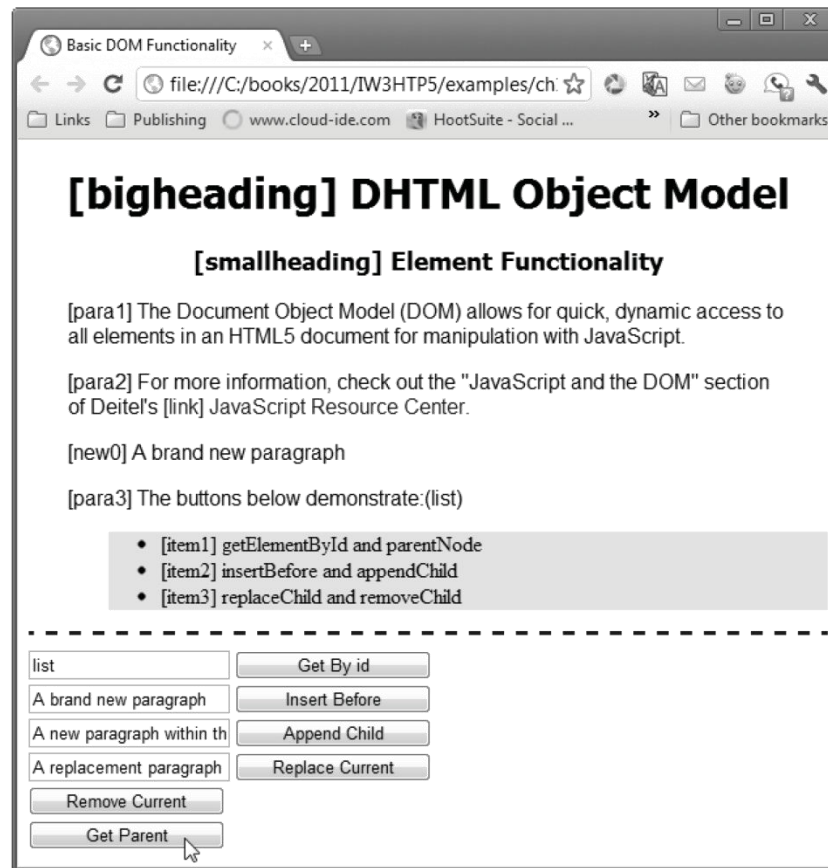


Fig. 12.11 | The document of Figure 12.4 after using the **Get By id** button to **item2**, then using the **Get Parent** button to select **item2**'s parent—the unordered list.



12.4 DOM Collections

- ▶ DOM has collections—groups of related objects on a page
- ▶ DOM collections are accessed as properties of DOM objects such as the document object or a DOM node
- ▶ The document object has properties containing the images collection, links collection, forms collection and anchors collection
 - Contain all the elements of the corresponding type on the page
- ▶ The collection's length property specifies the number of items in the collection



12.4 DOM Collections (Cont.)

- ▶ You access the elements of the collection using indices in square brackets
- ▶ `item` method of a DOM collection
 - An alternative to the square bracketed indices
 - Receives an integer argument and returns the corresponding item in the collection.
- ▶ `namedItem` method
 - receives an element `id` as an argument and finds the element with that `id` in the collection.
- ▶ `href` property of a DOM link node
 - Refers to the link's `href` attribute
- ▶ Collections allow easy access to all elements of a single type in a page
 - Useful for gathering elements into one place and for applying changes across an entire page

```
1  /* Fig. 12.12: style.css */
2  /* CSS for collections.html. */
3  body          { font-family: arial, helvetica, sans-serif }
4  h1            { font-family: tahoma, geneva, sans-serif;
5                text-align: center }
6  p a          { color: DarkRed }
7  ul            { font-size: .9em; }
8  li            { display: inline;
9                list-style-type: none;
10               border-right: 1px solid gray;
11               padding-left: 5px; padding-right: 5px; }
12 li:first-child { padding-left: 0px; }
13 li:last-child  { border-right: none; }
14 a              { text-decoration: none; }
15 a:hover        { text-decoration: underline; }
```

Fig. 12.12 | CSS for collections.html.



```
1  <!DOCTYPE html>
2
3  <!-- Fig. 12.13: collections.html -->
4  <!-- Using the links collection. -->
5  <html>
6      <head>
7          <meta charset="utf-8">
8          <title>Using Links Collection</title>
9          <link rel = "stylesheet" type = "text/css" href = "style.css">
10         <script src = "collections.js"></script>
11     </head>
12     <body>
13         <h1>Deitel Resource Centers</h1>
14         <p><a href = "http://www.deitel.com/">Deitel's website</a>
15             contains a growing
16             <a href = "http://www.deitel.com/ResourceCenters.html">list
17             of Resource Centers</a> on a wide range of topics. Many
18             Resource centers related to topics covered in this book,
19             <a href = "http://www.deitel.com/books/iw3http5">Internet &
20             World Wide Web How to Program, 5th Edition</a>. We have
21             Resource Centers on
22             <a href = "http://www.deitel.com/Web2.0">Web 2.0</a>,
23             <a href = "http://www.deitel.com/Firefox">Firefox</a> and
24             <a href = "http://www.deitel.com/IE9">Internet Explorer 9</a>,
```

Fig. 12.13 | Using the links collection. (Part 1 of 2.)



```
25         <a href = "http://www.deitel.com/HTML5">HTML5</a>, and
26         <a href = "http://www.deitel.com/JavaScript">JavaScript</a>.
27         Watch for related new Resource Centers.</p>
28         <p>Links in this page:</p>
29         <div id = "links"></div>
30     </body>
31 </html>
```

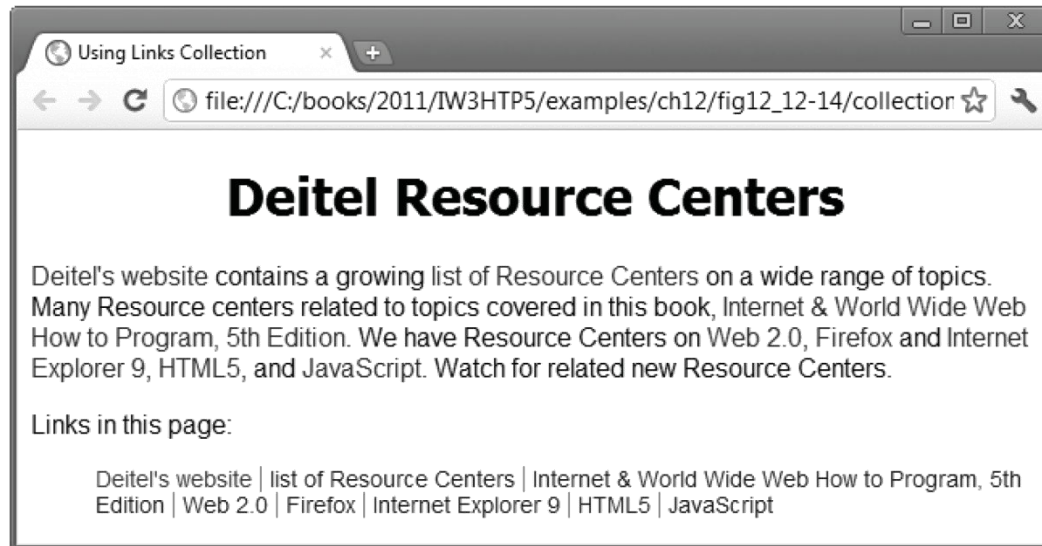


Fig. 12.13 | Using the 1inks collection. (Part 2 of 2.)



```
1 // Fig. 12.14: collections.js
2 // Script to demonstrate using the links collection.
3 function processLinks()
4 {
5     var linksList = document.links; // get the document's links
6     var contents = "<ul>";
7
8     // concatenate each link to contents
9     for ( var i = 0; i < linksList.length; ++i )
10    {
11        var currentLink = linksList[ i ];
12        contents += "<li><a href='" + currentLink.href + "'>" +
13            currentLink.innerHTML + "</li>";
14    } // end for
15
16    contents += "</ul>";
17    document.getElementById( "links" ).innerHTML = contents;
18 } // end function processLinks
19
20 window.addEventListener( "load", processLinks, false );
```

Fig. 12.14 | Script to demonstrate using the links collection.



12.5 Dynamic Styles

- ▶ An element's style can be changed dynamically
 - E.g., in response to user events
 - Can create mouse-hover effects, interactive menus and animations
- ▶ The document object's body property
 - Refers to the body element
- ▶ The `setAttribute` method is used to set the style attribute with the user-specified color for the `background-color` CSS property.
- ▶ If you have predefined CSS style classes defined for your document, you can also use the `setAttribute` method to set the `class` attribute.

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 12.15: dynamicstyle.html -->
4  <!-- Dynamic styles. -->
5  <html>
6      <head>
7          <meta charset="utf-8">
8          <title>Dynamic Styles</title>
9          <script src = "dynamicstyle.js"></script>
10     </head>
11     <body>
12         <p>Welcome to our website!</p>
13     </body>
14 </html>
```

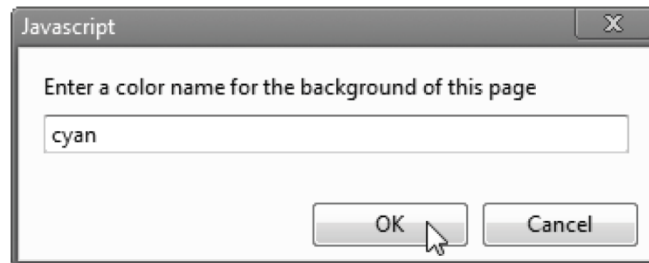


Fig. 12.15 | Dynamic styles. (Part I of 2.)

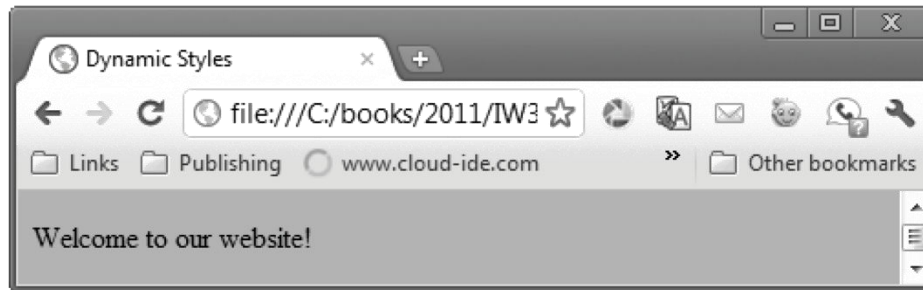


Fig. 12.15 | Dynamic styles. (Part 2 of 2.)



```
1 // Fig. 12.16: dynamicstyle.js
2 // Script to demonstrate dynamic styles.
3 function start()
4 {
5     var inputColor = prompt( "Enter a color name for the " +
6         "background of this page", "" );
7     document.body.setAttribute( "style",
8         "background-color: " + inputColor );
9 } // end function start
10
11 window.addEventListener( "load", start, false );
```

Fig. 12.16 | Script to demonstrate dynamic styles.

12.6 Using a Timer and Dynamic Styles to Create Animated Effects



- ▶ The next example introduces the window object's `setInterval` and `clearInterval` methods, combining them with dynamic styles to create animated effects.
- ▶ This example is a basic image viewer that allows you to select a book cover and view it in a larger size. When the user clicks a thumbnail image, the larger version grows from the top-left corner of the main image area.



```
1  /* Fig. 12.17: style.css */
2  /* CSS for coverviewer.html. */
3  #thumbs    { width: 192px;
4               height: 370px;
5               padding: 5px;
6               float: left }
7  #mainimg   { width: 289px;
8               padding: 5px;
9               float: left }
10 #imgCover   { height: 373px }
11 img        { border: 1px solid black }
```

Fig. 12.17 | CSS for coverviewer.html.



```
1  <!DOCTYPE html>
2
3  <!-- Fig. 12.18: coverviewer.html -->
4  <!-- Dynamic styles used for animation. -->
5  <html>
6      <head>
7          <meta charset = "utf-8">
8          <title>Deitel Book Cover Viewer</title>
9          <link rel = "stylesheet" type = "text/css" href = "style.css">
10         <script src = "coverviewer.js"></script>
11     </head>
```

Fig. 12.18 | Dynamic styles used for animation. (Part 1 of 6.)



```
12 <body>
13   <div id = "mainimg">
14     <img id = "imgCover" src = "fullsize/jhttp.jpg"
15       alt = "Full cover image">
16   </div>
17   <div id = "thumbs" >
18     <img src = "thumbs/jhttp.jpg" id = "jhttp"
19       alt = "Java How to Program cover">
20     <img src = "thumbs/iw3http.jpg" id = "iw3http"
21       alt = "Internet & World Wide Web How to Program cover">
22     <img src = "thumbs/cpphttp.jpg" id = "cpphttp"
23       alt = "C++ How to Program cover">
24     <img src = "thumbs/jhttplov.jpg" id = "jhttplov"
25       alt = "Java How to Program LOV cover">
26     <img src = "thumbs/cpphttplov.jpg" id = "cpphttplov"
27       alt = "C++ How to Program LOV cover">
28     <img src = "thumbs/vcsharphttp.jpg" id = "vcsharphttp"
29       alt = "Visual C# How to Program cover">
30   </div>
31 </body>
32 </html>
```

Fig. 12.18 | Dynamic styles used for animation. (Part 2 of 6.)

a) The cover viewer page loads with the cover of *Java How to Program*, 9/e



Fig. 12.18 | Dynamic styles used for animation. (Part 3 of 6.)

b) When the user clicks the thumbnail of *Internet & World Wide Web How to Program, 5/e*, the full-size image begins growing from the top-left corner of the window



Fig. 12.18 | Dynamic styles used for animation. (Part 4 of 6.)

c) The cover continues to grow

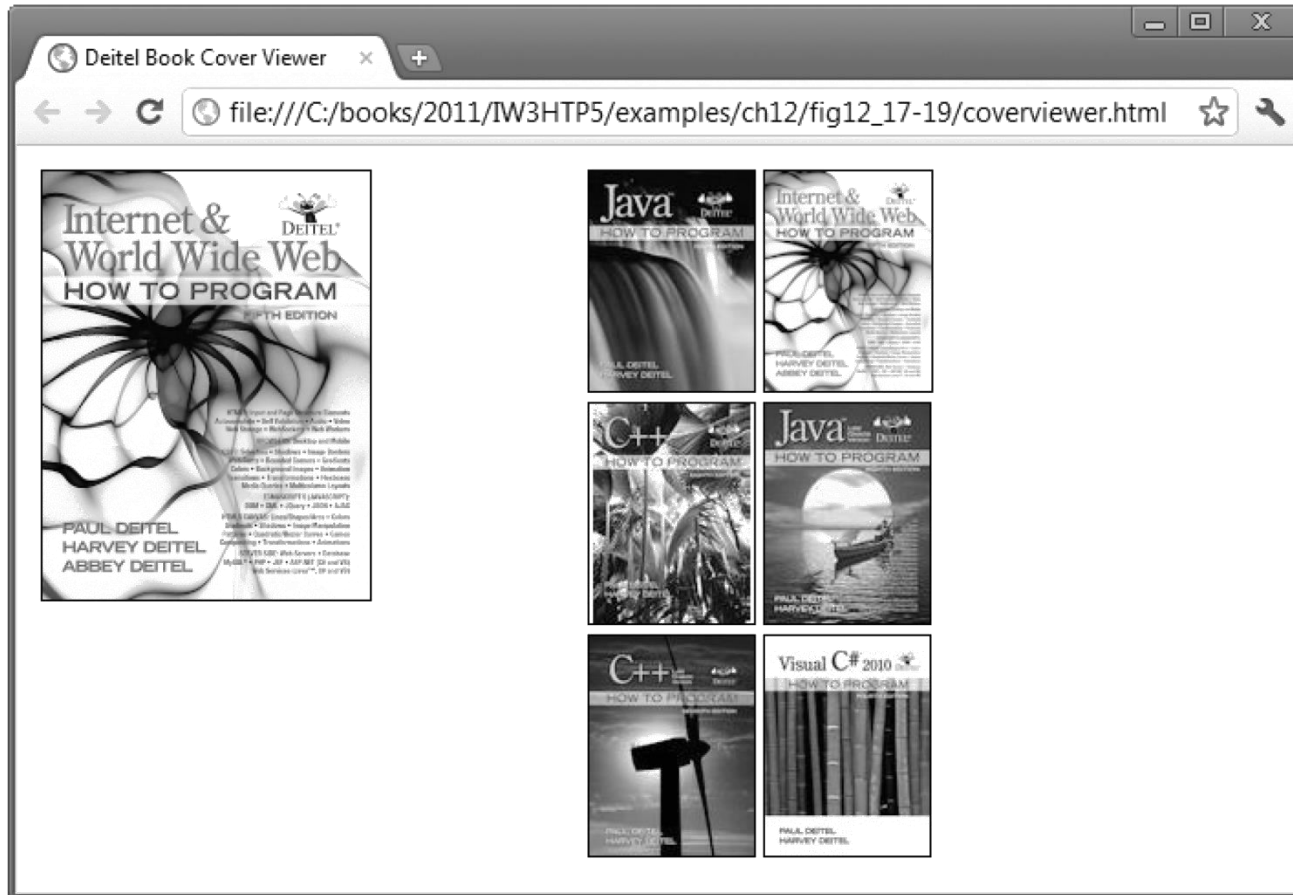


Fig. 12.18 | Dynamic styles used for animation. (Part 5 of 6.)

d) The animation finishes when the cover reaches its full size



Fig. 12.18 | Dynamic styles used for animation. (Part 6 of 6.)



```
1 // Fig. 12.19: coverviewer.js
2 // Script to demonstrate dynamic styles used for animation.
3 var interval = null; // keeps track of the interval
4 var speed = 6; // determines the speed of the animation
5 var count = 0; // size of the image during the animation
6
7 // called repeatedly to animate the book cover
8 function run()
9 {
10     count += speed;
11
12     // stop the animation when the image is large enough
13     if ( count >= 375 )
14     {
15         window.clearInterval( interval );
16         interval = null;
17     } // end if
18
19     var bigImage = document.getElementById( "imgCover" );
20     bigImage.setAttribute( "style", "width: " + (0.7656 * count + "px;") +
21         "height: " + (count + "px;") );
22 } // end function run
23
```

Fig. 12.19 | Script to demonstrate dynamic styles used for animation.
(Part I of 3.)



```
24 // inserts the proper image into the main image area and
25 // begins the animation
26 function display( imgfile )
27 {
28     if ( interval )
29         return;
30
31     var bigImage = document.getElementById( "imgCover" );
32     bigImage.setAttribute( "style", "width: 0px; height: 0px;" );
33     bigImage.setAttribute( "src", "fullsize/" + imgfile );
34     bigImage.setAttribute( "alt", "Large version of " + imgfile );
35     count = 0; // start the image at size 0
36     interval = window.setInterval( "run()", 10 ); // animate
37 } // end function display
```

Fig. 12.19 | Script to demonstrate dynamic styles used for animation.
(Part 2 of 3.)



```
38
39 // register event handlers
40 function start()
41 {
42     document.getElementById( "jhttp" ).addEventListener(
43         "click", function() { display( "jhttp.jpg" ); }, false );
44     document.getElementById( "iw3http" ).addEventListener(
45         "click", function() { display( "iw3http.jpg" ); }, false );
46     document.getElementById( "cpphttp" ).addEventListener(
47         "click", function() { display( "cpphttp.jpg" ); }, false );
48     document.getElementById( "jhttplov" ).addEventListener(
49         "click", function() { display( "jhttplov.jpg" ); }, false );
50     document.getElementById( "cpphttplov" ).addEventListener(
51         "click", function() { display( "cpphttplov.jpg" ); }, false );
52     document.getElementById( "vcsharphttp" ).addEventListener(
53         "click", function() { display( "vcsharphttp.jpg" ); }, false );
54 } // end function start
55
56 window.addEventListener( "load", start, false );
```

Fig. 12.19 | Script to demonstrate dynamic styles used for animation.
(Part 3 of 3.)

12.6 Using a Timer and Dynamic Styles to Create Animated Effects (Cont.)



- ▶ `setInterval` method of the window object
 - Repeatedly executes a statement on a certain interval
 - Takes two parameters
 - A statement to execute repeatedly
 - An integer specifying how often to execute it, in milliseconds
 - Returns a unique identifier to keep track of that particular interval.
- ▶ window object's `clearInterval` method
 - Stops the repetitive calls of object's `setInterval` method
 - Pass to `clearInterval` the interval identifier that `setInterval` returned
- ▶ Anonymous function
 - Defined with no name—it's created in nearly the same way as any other function, but with no identifier after the keyword function.